AD-A042 230 HAWAII UNIV HONOLULU F/G 9/2
AN INVESTIGATION OF COMPUTER SYSTEMS PROBLEMS.(U)
APR 77 W W PETERSON, A LEW DAAG29-74-G-0110
UNCLASSIFIED ARO-11944.16-M NL

END
DATE FILMED
8-77
DDC

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ARO-11944.16-M

AD A042230

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ARO-11944.16-M	2. GOVT ACCESSION NO.: ARO
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): AN INVESTIGATION OF COMPUTER SYSTEMS PROBLEMS	5. TYPE OF REPORT & PERIOD COVERED: Final Report: 1 Mar 74 — 28 Feb 77 15 Apr 77
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): W. W. Peterson, A. Lew	8. CONTRACT OR GRANT NUMBER(s): DAAG29-74-G-0110
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Hawaii, Honolulu, Hawaii 96822	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, Post Office Box 12211, Research Triangle Park, NC 27709	12. REPORT DATE: April 1977
	13. NUMBER OF PAGES: 12 28 p.
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DDC RECEIVED JUL 28 1977 E

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computers
Compilers
Diagnosis(general)
Debugging(computers)
Decision making
Tables(data)
Grammars
Optimization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The main computer system problem areas investigated were concerned with multiprogramming and paging systems, diagnostic compilers and debugging, and decision tables. Related areas investigated include Petri nets, grammatical inference, and mathematical optimization techniques.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

164500

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DDC FILE COPY

An Investigation of Computer Systems Problems

W.W. Peterson and A. Lew
Co-Principal Investigators

Department of Information and Computer Sciences
University of Hawaii at Manoa
2565 The Mall
Honolulu, Hawaii 96822

April 1977
Final Report

Prepared for the U.S. Army Research Office
under Grant No. DAAG29-74-G-0110, Project No. P-11944-EL

FOREWORD

This report is a Final Report summarizing research supported by the U.S. Army Research Office under Grant No. DAAG29-74-G-0110, Project No. P-11944-EL. This Final Report covers the entire period of the grant, from March 1, 1974 to April 15, 1977.

The main computer system problem areas investigated were concerned with multiprogramming and paging systems, diagnostic compilers and debugging, and decision tables. Related areas investigated include Petri nets, grammatical inference, and mathematical optimization techniques.

SUMMARY OF RESEARCH

The main problem areas investigated were multiprogramming and paging systems, diagnostic compilers and debugging, and decision tables.

In the area of multiprogramming and paging, we have studied the problems of optimal pagination (how to distribute program code and data into pages), replacement (how to decide which pages residing in main memory should next be removed), allotment (how to decide how many pages of each of several competing programs should be permitted in main memory), and scheduling (how to decide which programs should be active as a function of time). Of special note is the fact that, for a Markov program model, it is possible to calculate page fault costs for any given replacement algorithm and memory size. In principle, it is then possible to determine the best of a class of suboptimal replacement policies.

In the area of diagnostic compilers and debugging, we have studied on one hand the possibility of exploiting computer architectural features to facilate the debugging task, and on the other hand some implications of diagnostic objectives on computer architectural designs as well as on languages and programming methodology.

During the last half of the grant period, work on both of the two foregoing areas led to a consideration of decision tables--first, as an example of multiprocessing with redundancy constraints, and second, as an alternative to algorithmic languages with potential reliability advantages.

In addition, we have also studied a number of decidability questions related to Petri net models of concurrent processes, devised a new procedure for inferring grammars, and examined approximate methods for solving dynamic programming problems.

A. MULTIPROGRAMMING AND PAGING SYSTEMS

Structural Information for Virtual Memory Management [1,2,3]

We have examined methods of using a priori knowledge of program structure information for virtual memory management. A model of program behavior based upon localities was adopted (a locality is a set of pages which are used together in time), and a priori estimates of these localities can be obtained by using information about program connectivity. Then this structural information is used in new pagination, replacement, and memory allocation algorithms [1,2].

For the pagination problem, a primary shortcoming in most earlier solutions is the assumption of single page localities implicitly made in adopting the criterion of minimizing the number of interpage references. We have instead adopted a space-time cost criterion which includes the cost of references made outside the resident set (the current locality), as well as the cost of main memory occupied at any time. We amend another major shortcoming by treating data as a class separate from instructions. A locality then consists of instruction and data sublocalities.

For segregated pagination it would be desirable to distribute data such that all the data required by an instruction sublocality is grouped together. Of course, this cannot be satisfied for all instruction sublocalities, since data may be common to different instruction sublocalities. However, paginating data nodes so as to minimize the total number of pages (of data) possibly referenced by each instruction sublocality, so that a maximum number of data pages can be kept in main memory for a given resident set size, can be done using just structural information. If inter-locality reference frequencies can be estimated, the pagination can be carried out to minimize an expected space-time cost.

With regard to demand-paging replacement, we have shown how optimal page

replacement decisions can be determined, by means of dynamic programming, utilizing page structure information. (This is based on a stochastic control process model of demand-paging systems [3]). We have also proposed two new "optimal" algorithms, optimal in the sense of minimizing the expected number of page faults, under Markovian and locality assumptions on program execution. LPR(N), based upon knowledge of transition probabilities among pages, replaces a page with the Least Probability of Reference along all paths of length < N from the currently referenced page. This may be regarded as a generalization of LTP [4], which would replace that page with Least Transition Probability (i.e. least probability of next reference). INR, based upon structural rather than probabilistic information, replaces a page which has the maximum Interval (path length) of Next Reference from the currently referenced page. INR replacement may be regarded as a realizable approximation to Belady's unrealizable optimal algorithm. Both schemes would replace a page with "furthest" next reference, except in case of INR this information is normally a conservative estimate. The maximum value of INR obtained from structural information guarantees that the page will not be referenced before that interval. INR may also be contrasted with LNR [4], which would replace that page with longest expected next reference (based upon probabilistic rather than structural information).

Exemptive replacement algorithms divide the resident set into an exempt set, e.g. the set of pages which have a path of length < N from the currently referenced page, and a nonexempt set. The page selected to be replaced is the lowest ranking, based upon conventional reference history information, page in the latter set. Conditions under which exemptive replacement is provably better (in terms of page fault costs)

than the underlying historical algorithm were studied: the main condition is that a page exempted from replacement must not rise in ranking unless it is referenced.

Memory allocation is divided into three sub-problems: allotment, locality size estimation, and locality membership determination. For allotment, i.e. the division of main memory resources among programs, we have shown under probabilistic assumptions that for non-identical programs an "optimal" allotment must at least equal the locality size estimate and that "excess" memory must be distributed in proportion to variances in locality size. To estimate locality size, we have employed Bayesian decision techniques to combine structural and historical information. An a priori estimate can be made based upon page connectivity, and posteriori estimates can be made based upon reference-history samples. Finally, the problem of memory membership, i.e. determination of what pages to keep as members of the locality, is considered for a pre-loading environment. (For demand paging, the choice of a replacement algorithm determines the loading strategy.) A priori information on localities is used to load an entire locality as a batch at the time it is entered. This saves a large number of page faults which would occur if the pages of a locality were loaded separately.

Results for Markov Paging Models [5,6]

Given a Markov chain model of paging programs for any given (realizable) replacement algorithm and memory size (page allotment), the page fault probability can be calculated by standard statistical techniques [7,8]. Furthermore, if the replacement problem is formulated as a Markovian decision process, Howard's policy iteration algorithm can be used to

find an optimal replacement policy [7,9]. These two concepts provide the basis for the solution of several unresolved optimal paging problems for a page-fault cost criterion.

Prior solutions to the optimal pagination problem are unrealistic because they do not take into account environmental (run-time) considerations -- e.g. replacement policies and page allotment. The ability to calculate page fault probabilities for any given replacement policy and page allotment provides the basis for determining optimal paginations with respect to these environmental factors. Since there are only a finite number of paginations possible for any program, the optimal one can therefore be determined, in principle in not in practice, if in no other way than by enumerative methods.

A formal solution to the optimal allotment (partitioning) problem requires a quantitative specification of criterion functions $C_k(m)$, the cost of allotting m page frames to program k of size n_k. The optimal set of page allotments minimizes the sum $\Sigma_k C_k(m_k)$ subject to space limitations and various multiprogramming or scheduling constraints. The ability to calculate page fault probabilities for any given replacement policy and page allotment provides the basis for solving this optimization problem (for a page-fault cost criterion). We emphasize that the replacement policy is a parameter.

Howard's policy iteration procedure may be used to determine the best of given classes of replacement policies (e.g. where decisions depend only upon the program state, memory state, or next referenced page). Each class of policies may be associated with a constraint on the associated Markovian decision process. One class of constraint, implicitly adopted heretofore, is where replacement decisions may not depend upon program

structure information. "Exemptive" algorithms, as proposed before, relaxes this constraint. The ability to calculate page fault probabilities permits the value of an exemption based upon a given page connection to be determined by comparing page-fault costs with and without use of the exemption, thereby reducing run-time overhead for checking connectivity information that does not yield a counter-balancing improvement in performance.

Optimal Resource Allocation Among Parallel Processes [10]

Optimal allocation of resources (e.g. page frames) among competing parallel processes has been formulated as a problem in finding the shortest path (or route) in a directed graph. This formulation requires a quantitative measure of the performance of each process as a function of its resource allocation: for $k = 1,\ldots,N$, let $C_k(q_k)$ denote a non-negative real measure of the performance of process P_k given an allocation of integral size q_k. The routing problem can be solved, for example, by dynamic programming.

In general, $C_k(q_k)$ may be a time-varying function, $C_k(q_k,t)$, in which case the routing problem may be re-solved "every so often" yielding time-varying allocations $q_k^*(t)$. The discrete times at which reallocations may be made can depend upon time-slices, process blockings, monitored process performance measures, or a priori information.

A reformulation of the problem to take into account reallocation costs (in part to reflect the overhead required to change the amount of resources allocated to any process, and in part to reflect the calculations required to re-solve the routing problem) has been made. Furthermore, by allowing $q_k(t)$ to be zero, our formulation automatically yields the optimal

"degree of multiprogramming" (number of active processes). After solving for the optimal set of allocations, $\{q_k^*(t) \mid k = 1,\ldots,N\}$, the optimal degree of multiprogramming is simply the number of nonzero $q_k^*(t)$.

If a priori (statistical) knowledge of $C_k(q_k, t + \Delta)$, $\Delta > 0$, is available, the allocation problem may be formulated as a (Markovian) multistage decision process. An optimal sequence of allocations (i.e. "schedule") in this context must then minimize the sum of the allocation costs over time. Dynamic programming is generalizable to this class of problems.

When parallel processes are not independent, so that they interfere if executed concurrently [11], allocation decisions should take into account the dependencies. For example, if processes are partially ordered according to a temporal precedence relation, for (say) determinacy reasons, then resources should be allocated at any point in time only to "eligible" processes whose predecessors have all been completed. If (cyclic) processes must satisfy synchronization or exclusion constraints, again only certain processes are "eligible" for resources. Formally, we may set the allocation costs for ineligible processes arbitrarily high, and use the foregoing procedures to distribute resources among the eligible processes.

When allocation of resources is constrained by "soft" precedence relations among processes, where there may be preferred rather than absolute precedences (superseding other measures of allocation cost), a problem of a different nature arises. We may in this case associate a higher cost with one ordering of two processes than another, where we assume these costs are independent of the sizes of the allocations. Then the optimal ordering of the processes can be found by solving a

"traveling-salesperson" routing problem--e.g. by dynamic programming.

The situation may arise in parallel processing applications where a set of processes has "redundancies." A process, for example, may no longer be necessary depending upon the outcomes of certain other processes, so that it would be wasteful to execute the former process first. If decisions as to whether or not to execute processes depend upon others, the problem of determining an optimal hierarchy (tree) of process executions arises. This problem has also been solved by means of dynamic programming.

If we assume nonpreemptible resources, or more weakly that there is a nonzero lower bound on the amount of resources a process can be allocated once activated (possibly a time-varying bound), then requests for additional resources by processes cannot be granted on demand without risk of a "deadlock" [11]. The shortest path may then no longer be an acceptable solution to the optimal resource allocation problem. In this event, a "safety" constraint may be formally added to the optimization problem.

An alternative approach is the direct utilization of dynamic programming to determine the k-th shortest path, for k = 1, 2,.... The smallest k for which the k-th shortest path does not violate the safety criterion then yields the desired solution. We note that if no such path exists, the system is initially deadlocked. In this event, one or more of the deadlocked processes must be "aborted" to effect a recovery, at a cost which should be minimized.

One method of preventing deadlocks where there are different types of resources is to require that processes use the resources in a fixed order. Given this ordering of resources, the problem of optimally ordering processes is a "flow-shop" problem, solvable by means of dynamic programming for two resource types.

References

[1] Jain, N., "A Note on Use of Structural a Priori Information for Virtual Memory Management"

[2] Jain, N., Use of Program Structure Information in Virtual Memory Management

[3] Lew, A., "Optimal Control of Demand-Paging Systems"

[4] Mattson, R.L., et al., "Evaluation Techniques for Storage Hierarchies", IBM Syst. J., 1970, pp. 78-117.

[5] Lew, A., "Some Remarks on Optimal Paging Algorithms"

[6] Lew, A., "On Optimal Demand-Paging Algorithms"

[7] Ingargiola, G. and Korsh, J.F., "Finding Optimal Demand-Paging Algorithms," JACM, 1974, pp. 40-53.

[8] Franklin, M.S. and Gupta, R.K., "Computation of Page Fault Probability from Program Transition Diagram," Comm. ACM, 1974, pp. 186-191.

[9] Kogan, Y.A., "Markov Models of Page Replacement in a Two-Level (Virtual) Computer Memory," Autom. and Rem. Ctol., 1973, pp. 641-648.

[10] Lew, A., "Optimal Resource Allocation and Scheduling Among Parallel Processes"

[11] Coffman, E.G., Jr. and Denning, P.J., Operating Systems Theory, Prentice-Hall, 1973.

B. DIAGNOSTIC COMPILERS AND DEBUGGING

Microarchitectures and Debugging [1,2,3]

Microprocessor implementations of compilers and associated run-time environments (e.g. "pipelining") have long been suggested for efficiency reasons. It is reasonable to expect then that many methods for improving diagnostic capabilities that are impractical to implement in software will turn out to be practical if implemented in hardware or firmware. Therefore we have investigated computer architectural features that would facilitate the debugging task. "Tagged" architectures [4] may be used for implementing traps, for example. However, we have limited our study primarily to microprocessor designs. While microprograms for diagnosing hardware errors have long been used and remain of continuing interest, little has been done at the microprogramming level for diagnosing user program software errors.

Suitable microprocessor architectures can aid syntactic error detection and correction. If microprograms are used to perform context checks, then parsing approaches previously discounted as too inefficient, but which have diagnostic advantages, may come to the fore. A microprocessor which permits associative searching of substrings would be particularly desirable.

Many semantic errors can be detected at run-time because an operational "violation" (fault) will occur. Faults can be detected by means of hardware (e.g. division by zero) or software (e.g. subscript bound) traps, or some combination of the two (cf. test for use of undefined variables). Certain traps may only be practical (if at all) at the microprogramming level; an example is that of bounding values of variables. (The bounds can be supplied in compile-time declarations, as analog and COBOL programmers must typically make.) With the availability of "dynamic user microprogramming"

[5], user trap commands can be a particularly valuable tool.

A trap generally detects a symptom of an error, not its root cause. Extensive post-mortem analyses would generally be necessary to identify causes. To locate references to a suspicious variable, an associative processor would be of great assistance. A search through program memory would not be necessary if compile-time information (e.g. instruction and data cross-reference tables) were saved for run-time reference. Micro-architectures that would facilitate representation and processing of such structural information (e.g. as sparse matrices) would be helpful.

Tracing of control flow or data accesses by insertion of probes has been suggested earlier. Implementation of these probes at the micro-programming level is a natural idea. While boolean probes can provide important information, "frequency" probes are even more useful. A record of flow and data access history is particularly of value, but requires much memory space (generally too much for micro-control storage). Therefore microarchitectures facilitating the use of noncontrol memory (main memory [5] or perhaps a dedicated buffer memory) for storage of monitored information are desirable.

Finally, interactive graphics systems have a significant influence on the feasibility of diagnostic approaches. Hence microarchitectures designed to facilitate graphics displays should prove of immense value.

Decision Table Programming [6]

Decision tables are commonly thought to be restricted in applicability to procedures involving sequencing of tests, nested-IFs, or CASE statements. However, we have shown that a decision table can implement any computable function. One need only observe that any Turing Machine program can be

"emulated" by a decision table by letting each Turing Machine instruction of the form (input,state)→(output,tape movement,state) be represented by a decision table rule where (input,state) are conditions and (output, tape movement,state) are actions.

From a more practical point of view, it can also be shown that all computer program flowcharts can be emulated by a decision table. That this is true follows from the reducibility of all flowcharts to, in essence, CASE statement form: generally, this requires the introduction of a special variable for sequencing. Decision tables of this nature are not recommended, however, since the underlying program logic is obscured. Furthermore, emulation of a "poor" flowchart is likely to lead to a poor decision table.

The foregoing means, in particular, that decision tables are applicable to all scientific problems as well as to more traditional business data processing problems. In fact, we may regard decision tables as a general-purpose programming language with a control structure that enforces certain desirable disciplines. To gain wider acceptance, the development of efficient translator/processors and of effective automated decision-table-program evaluation tools will be necessary. We have worked towards these goals, first by developing an optimizing decision-table compiler procedure (discussed in Section C), and second by showing how numerous software reliability concepts are applicable to, and in some cases facilitated by, the use of decision tables.

For example, we have found that GOTO actions can be eliminated from decision tables, essentially by use of a sequencing variable. An algorithm for "reducing" decision tables, subject to certain restrictions on condition and action dependencies, was also developed; a reduction of

a decision table is a transformation of the table to an equivalent one making less use of an auxiliary variable for sequencing.

We have also shown how checks for condition dependencies or excludable rules can be made in theory. In practice, these checks are feasible only under certain restrictive assumptions.

The tabular form of decision tables offers certain advantages as far as "systematizing" correctness proofs is concerned. However, we are not yet prepared to claim that decision tables are significantly easier to prove correct. Difficulties relating to finding loop invariants, for example, remain.

References

[1] Lew, A., " A BASIC Operating System and Interpreter for Diagnostic and Pedagogic Purposes"

[2] Lew, A., "Diagnostic Compilers and Debugging"

[3] Lew, A., "Diagnostic Compiler and Debugging Systems: Microarchitectural Implications"

[4] Feustel, E.A., "On the Advantages of Tagged Architecture," IEEE Trans. Comput., 1973, pp. 644-656.

[5] Thomas, R.T., "Organization for Execution of User Microprograms from Main Memory: Synthesis and Analysis," IEEE Trans. Comput., 1974, pp. 733-790.

[6] Lew, A. and Tamanaha, D., "Decision Table Programming and Reliability"

C. DECISION TABLE CONVERSION [1]

Decision tables have long been an important tool, with applicability to numerous practical business problems (e.g. work flow, product design, quality control, and manufacturing). Above, we have indicated that they are applicable to all scientific problems as well. The desire for efficient run-time processing of decision tables has led to many algorithms for converting limited-entry decision tables to "optimal" computer programs, most of which are heuristic. Two algorithms, however, guarantee optimality: that of Reinwald-Soland [2,3] and Bayes [4]. The former is based upon a branch-and-bound technique and the latter upon dynamic programming. The two are related, as are all mathematical programming procedures. They guarantee optimality because they are essentially enumerative (combinatorial) algorithms; no nonenumerative algorithm can be optimal in general. The price of run-time optimality is, of course, compile-time computational complexity.

We have extended the works of Reinwald-Soland, and Bayes, and more recently of Schumacher-Sevcik [5], to general decision tables. Specifically, we developed a dynamic programming algorithm which guarantees optimality, taking into account extended or mixed entries, rule frequencies, different optimality criteria, common action sets, ELSE rules, sequencing constraints on condition tests, and excludable combinations of conditions. Transformation of extended-entry to limited-entry decision tables prior to application of a conversion algorithm has been shown to introduce inefficiencies; among other problems, the cost of testing conditions may not be separable.

Given a set of condition-variables $\{X_1,\ldots,X_N\}$, let $V_i = \{v_{i1},\ldots,v_{ir_i}\}$ be a set of "outcomes" (extended-entries) associated with X_i. Testing of X_i selects one element of V_i as its outcome, say v_{ij}; we denote this by

writing $e(X_i) = v_{ij}$. Let $C(X_i|R)$ be the cost of testing X_i given that the predicate R is true. Define $f_k(\{X_{s_1}, X_{s_2}, \ldots, X_{s_k}\}|R_k)$ as the minimum achievable cost having the k remaining untested variables $\{X_{s_1}, \ldots, X_{s_k}\}$ given that R_k is true. Then, applying dynamic programming, we have the following: Proposition. The optimal order of testing can be found by solving

$$f_k(\{X_1,\ldots,X_{s_k}\}|R_k) = \min_i \{C(X_i|R_k) + \sum_{j=1}^{r_i} f_{k-1}(\{X_{s_1},\ldots,X_{s_k}\} - \{X_i\}|R_k \wedge e(X_i) = v_{ij}\}$$

for $f_N(\{X_1,\ldots,X_N\}|.\text{TRUE}.)$ given that $f_1(\{X_i\}|R_1) = C(X_i|R_1)$. The $i \in \{s_1,\ldots,s_k\}$ which yields the minimum in the above functional equation designates the variable to be tested next (at stage k; k=N,...,1) given that R_k is true. (We remark that this proposition stemmed from our earlier study of redundant parallel processes [6].)

We observe that R_k is the intersection of predicates of the form $e(X_i) = v_{ij}$ for X_i not in the set $\{X_{s_1},\ldots,X_{s_k}\}$ of variables tested at the earlier stages, k+1,...,N. If $\{X_{s_1},\ldots,X_{s_k}\}$ consists only of variables which need not be tested given the outcome R_k, then $C(X_i|R_k) = 0$ for each such variable. Otherwise, we may let $C(X_i|R_k) = c_i * \Pi(R_k)$, where c_i is the cost of testing X_i, and $\Pi(R_k)$ is the probability that R_k occurs. (The latter depends upon likelihoods of various outcomes; equi-likelihood may be assumed if probabilistic or rule-frequency information is not available.)

With the above definition our dynamic programming procedure yields the hierarchical order of testing which minimizes the expected total cost associated with the decision table. It should be emphasized that the ordering is not linear, but is of tree form.

To minimize expected time, let c_i be the time required to test condition-variable X_i, and $\Pi(R_k)$ be the probability that the specified combination of outcomes R_k obtains. $C(X_i|R_k) = 0$ if the subtable associated with R_k has a common action set.

To minimize space, let c_i be the space required to test X_i, and $\Pi(R_k) = 1$ if the probability of R_k is nonzero, else $\Pi(R_k) = 0$.

Sequencing constraints of the type X_p must be tested before X_q can be handled by restricting the minimum in the functional equation to be taken over only those X_i in $\{X_{s_1}, X_{s_2}, \ldots, X_{s_k}\}$ which can be tested before each of the remaining condition variables. We note that each such constraint decreases the computational requirements of the algorithm. More complex constraints can be handled in analogous fashion.

An ELSE rule can in principle be handled by replacing it by an equivalent set of equi-probable rules with a common action. These rules may then be treated no differently from other rules. A rule with a dashed ("don't-care") entry can also be replaced by an equivalent set of equi-probable rules, one for each predicate associated with the corresponding condition test. In both cases, the introduced rules need not be assumed equi-probable if additional information is known.

Excludable rules are those with zero probability of occurrence. Tests for such rules are unnecessary in principle, although for error-detection reasons they may be desired nevertheless. An ELSE-rule is frequently used as a catch-all for such a purpose. The dynamic programming algorithm as given above (as well as the Reinwald-Soland algorithm) will yield tests for excludable rules. In the event these tests are not wanted, a lower cost solution can be found using the dynamic programming algorithm by setting $\Pi(R)$ equal to zero if the set of actions associated with the truth of R, exclusive

of actions of excludable rules, does not contain more than one element.

Furthermore, we observe that, insofar as our extension to extended-entries is concerned, we have assumed a single cost for testing a condition for all its possible outcomes. Where this cost can be divided among the different outcomes, a division of the extended-entry condition into several others (not necessarily limited-entry ones) would lead, in general, to a better solution.

References

[1] Lew, A., "Optimal Conversion of Extended-Entry Decision Tables with General Cost Criteria"

[2] Reinwald, L.T. and Soland, R.M., "Conversion of Limited-Entry Decision Tables to Optimal Computer Programs I," JACM, 1966, pp. 339-358.

[3] Reinwald, L.T. and Soland, R.M., "Conversion of Limited-Entry Decision Tables to Optimal Computer Programs II," JACM, 1967, pp. 742-756.

[4] Bayes, A.J., "A Dynamic Programming Algorithm to Optimize Decision Table Code," Australian Comp. J., 1973, pp. 77-79.

[5] Schumacher, H. and Sevcik, K.C., "The Synthetic Approach to Decision Table Conversion," Comm.ACM, 1976, pp. 343-351.

[6] Lew, A., "Optimal Resource Allocation and Scheduling Among Parallel Processes"

D. DECIDABILITY PROBLEMS AND PETRI NETS [1,2,3]

We have considered a version of vector addition systems [4] for representing concurrent processes that uses the synchronization primitive PV (due to Dijkstra) or its generalizations. A class of languages which can be specified as the sets of allowable operation sequences of vector addition systems has been studied. We have shown that it is decidable for a linear functional δ on the reachable configurations whether δ takes only a finite number of values over the reachable configurations, and the decision problem whether δ is lower-bounded by a given constant has been shown to be equivalent to the point-reachability problem for vector addition systems. The decision problem whether a vector addition system is deadlock-free (or live) has been shown to be equivalent to the point-reachability problem. A standard technique for proving correctness of programs with the synchronization primitive PV or its generalizations has also been formulated based on Presburger logic. Some problems on how to find a vector addition system which meets specified synchronization requirements expressed as a Presburger sentence have been studied.

Vector-addition-system can be represented as Petri nets [5]. Petri-net models of synchronized concurrent processes were also studied, and a number of other decidability results were established. The "reachability" problem (i.e. whether or not an arbitrary "state" can be entered) is an open problem, and we have shown a number of other problems to be equivalent: namely, "liveness", "deadlock-freeness", "consistency", "reversibility", "equality", and "inclusion." Some special conditions, under which the reachability problem is undecidable, were also found.

Finally, suppose that a concurrent system is represented as a Petri net N with an initial marking M which represents an initial safe state of the

system. Let $R_N(M)$ be the set of markings reachable from N, and let $C_N(M)$ be the set of markings M' such that $M' \in R_N(M)$ and $M \in R_N(M)$. If $R_N(M) = C_N(M)$, then N is said to be M-reversible and deadlocks are said to be prevented in the system. We have shown that the deadlock prevention problem in this sense is decidable. Deadlocks are said to be avoided in the system if and only if actions can be chosen in such a way that only markings in $C_N(M)$ are reachable. We have also shown that the deadlock avoidance problem in this sense is decidable. Furthermore, Petri nets in which deadlocks are prevented and which have the same reachability set as that of a given deadlock-avoiding system can be constructed.

References

[1] Kasami, T., Vector Addition Systems and Synchronization Problems of Concurrent Processes, University of Hawaii, memo ARO-23, Sept. 1974.

[2] Araki, T. and Kasami, T., "Some Decision Problems Related to the Reachability Problem for Petri Nets"

[3] Araki, T. and Kasami, T., "Decidable Problems on the Strong Connectivity of Petri Net Reachability Sets"

[4] Karp, R.M. and Miller, R.E., "Parallel Program-Schemata," JCSS, 1969, pp. 147-195.

[5] Hack, M. Decision Problems for Petri Nets and Vector Addition Systems, MIT, memo 95-1, August 1974.

E. GRAMMATICAL INFERENCE SCHEMES

The grammatical inference problem is the task of specifying a grammar that "best" describes a particular set of character strings. The associated language should contain all the strings in the set and it should enjoy all the properties that normally might be inferred from the sample set. Results in this area are likely to have some impact on such areas as model building, concept formulation and decision making under uncertainty.

Much of the original work on this subject was done by Feldman, et al. [1,2,3]. Their first cut at viable programs for inferring regular and pivot grammars is described in [1]. This pioneering work has served to delimit some of the major avenues of research in this area and to stimulate others to explore related areas [4,5]. In particular we have been investigating the extension of this problem to the case where input/output strings are considered. The corresponding transducer inference problem seems to be a natural setting for the traditional estimation problems encountered in systems theory.

In terms of actual inference schemes, we have made one major modification to that which was reported in [1], where a very concerted effort was made to derive grammars that generated infinite sets of strings. In our procedure, the manner in which production rules are combined and simplified produce this effect as a natural consequence of the nature of the data. No explicit attempt is made to force a recursive grammar. Another modification is that our scheme is able to process input/output pairs in any order. This flexibility allows the scheme to be used interactively so that additional inputs based on intermediate results can be supplied. Our scheme has been implemented in SNOBOL.

A comparison of some of our results with those reported in [1]

indicate that in general our scheme was able to infer the "correct" model with the same or fewer samples. There were two cases in which the inferred models differed significantly. In one case we were able to infer an infinite language where their model produced a finite language. The number of nonterminals in both models were the same and the nature of the productions were similar. This was a rather surprising result in view of the fact that our scheme does not intentionally go after recursive grammars. In the other case we inferred a simpler model (requiring one less nonterminal and similar production rules).

Currently we are investigating two avenues of further research. One is concerned with the manner in which this program is used to function interactively with a user. Since the inference scheme is sensitive to the arrangement of the sampled data, we would like to find good empirical ways of selecting new input samples based on the results for previous data. The other area of interest is concerned with finding upper bounds on the lengths of the sample input strings based on the model complexities that we are willing to tolerate. This would be the first step in a trade-off analysis between grammatical complexity and modelling accuracy.

References

[1] Feldman, J.A., et al., Grammatical Complexity and Inference, Stanford, memo AI-89, June 1969.

[2] Feldman, J., "Some Decidability Results on Grammatical Inference and Complexity," Information and Control, pp. 244-262, 1972.

[3] Horning, J.J., A Study of Grammatical Inference, Stanford, memo AI-98, August 1969.

[4] Fu, K. and Booth, T.L., "Grammatical Inference: Introduction and Survey - Part I," IEEE Trans. on Systems, Man, and Cybernetics, January 1975.

[5] Fu, K. and Booth, T.L., "Grammatical Inference: Introduction and Survey - Part II," IEEE Trans. on Systems, Man and Cybernetics, July 1975.

F. APPROXIMATE DYNAMIC PROGRAMMING [1]

Mathematical programming techniques have frequently been employed in the solution of computer systems optimization problems. For example, dynamic programming was used for multiprogramming problems [2,3] and for decision table conversion [4]. Unfortunately, for large systems, determination of optimal solutions may become impractical for combinatorial reasons. Consequently, approximation techniques may be necessary.

We have continued earlier studies on approximation methods made in the context of "control" problems [5] in the hope that generalizations to computer systems problems would be possible. While the precise results obtained do not appear applicable, the general philosophy adopted---that approximate methods for solving complex problems may (but not necessarily) be better than exact methods for solving oversimplified problems---does appear applicable. We have not, however, pursued this.

References

[1] Lew, A., "Stability, Prediction-Correction, and Dynamic Programming"

[2] Lew, A., "Optimal Resource Allocation and Scheduling Among Parallel Processes"

[3] Lew, A., "Optimal Control of Demand-Paging Systems"

[4] Lew, A., "Optimal Conversion fo Extended-Entry Decision Table with General Cost Criteria"

[5] Lew, A., "A Predictor-Corrector Method for Dynamic Programming," IEEE Transactions on Automatic Control, 1974, pp. 54-56.

LIST OF PUBLICATIONS

1. Lew, A., "Optimal Resource Allocation and Scheduling Among Parallel Processes," in Parallel Processing, ed. T. Feng, Springer-Verlag, Berlin, 1975, pp. 187-202.

2. Jain, N., "A Note on Use of Structural a Priori Information for Virtual Memory Management," Proc. HICSS-8, January 1975, pp. 129-132.

3. Lew, A., "Some Remarks on Optimal Paging Algorithms," Proc. HICSS-8, January 1975, p. 133.

4. Jain, N., Use of Program Structure Information in Virtual Memory Management, Ph.D. Dissertation, University of Hawaii, April 1975.

5. Lew, A., "A BASIC Operating System and Interpreter for Diagnostic and Pedagogic Purposes," in Microarchitecture of Computer Systems, ed. R. W. Hartenstein and R. Zaks, North-Holland, Amsterdam, 1975, pp. 85-91.

6. Lew, A., "On Optimal Demand-Paging Algorithms," Proc. 2nd USA-Japan Comp. Conf., August 1975, pp. 596-600.

7. Lew, A., "Diagnostic Compilers and Debugging," Proc. Internatl. Comp. Symp., Vol. II, August 1975, pp. 205-212.

8. Lew, A., "Diagnostic Compiler and Debugging Systems: Microarchitectural Implications," Proc. 4th Texas Conf. Comp. Sys., November 1975, pp. (2A-1) 1-3.

9. Lew, A., "Stability, Prediction-Correction, and Dynamic Programming," J. Optim. Th. and Applics., Vol. 17, Nos. 3/4, November 1975, pp. 239-250.

10. Lew, A., "Optimal Control of Demand-Paging Systems," Information Sciences, Vol. 10, No. 4, 1976, pp. 319-330.

11. Lew, A. and Tamanaha, D., "Decision Table Programming and Reliability," Proc. 2nd Internatl. Conf. on Software Engrg., October 1976, pp. 345-349.

12. Araki, T. and Kasami, T., "Some Decision Problems Related to the Reachability Problem for Petri Nets," Theoretical Computer Science, Vol. 3, No. 1, 1977, pp. 85-104.

13. Araki, T. and Kasami, T., "Decidable Problems on the Strong Connectivity of Petri Net Reachability Sets," Theoretical Computer Science, to appear.

14. Lew, A., "Optimal Conversion of Extended-Entry Decision Tables with General Cost Criteria," Commun. ACM, to appear.

LIST OF PARTICIPATING PERSONNEL

1. W. W. Peterson, Co-Principal Investigator
2. A. Lew, Co-Principal Investigator
3. T. Kasami, Research Associate
4. S. Y. Itoga, Research Associate
5. N. Jain, Graduate Assistant, awarded Ph.D. degree, May 1975.
6. W. Pang, Graduate Assistant, awarded M.S. degree, May 1975.
7. D. Tamanaha, Graduate Assistant
8. K. Kajiwara, Graduate Assistant